1.Wstęp

Celem pracy jest przeprowadzenie badania dotyczącego różnych zastosowań metodyki Scrum

w organizacjach. Badania zostały przeprowadzone w formie ankiety udostępnianej za pomocą mediów społecznościowych, grup tematycznych oraz bezpośrednich wywiadów.

Zakres pracy obejmuje badanie literatury, przegląd źródeł internetowych, przegląd czasopism dziedzinowych. Określenie celu i zakresu badań. Zaprojektowanie badania i wybór metody badawczej. Wypracowanie wniosków i propozycji dobrych i złych praktyk w stosowaniu metodyki Scrum.

Narzędziem zastosowanym do przeprowadzenia ankiety był Google Form, a jako narzędzie do analizy i stworzenia wizualizacji użyty został MS Excel.

Rezultatem pracy jest odpowiedź na pytanie jaki procent zespołów korzysta z metodyki Scrum poprawnie oraz gdzie najczęściej wprowadzane są zmiany i jakie.

Praca składa się z pięciu części. Rozdział pierwszy jest wprowadzeniem do tematyki pracy.

W drugim rozdziale opisane są metodyki zwinne wraz z opisem kilku poszczególnych przykładów.

W trzecim rozdziale przedstawiona jest metodyka Scrum na podstawie analizy Scrum Guide. W czwartym rozdziale opisane jest badanie dotyczące analizy różnych adaptacji metodyki Scrum w zespołach wytwórczych. W piątym i ostatnim rozdziale zostały przedstawione wnioski wraz z opisem dobrych i złych praktyk w zastosowaniu metodyki Scrum. Na końcu pracy został dołączony również słownik,

z tłumaczeniem anglikanizmów i wyjaśnieniem pojęć oraz Bibliografia.

2. Wprowadzenie

Od momentu powstania dziedziny wytwarzania oprogramowania pojawiła się potrzeba odpowiedniego zarządzania projektami IT. Od samego początku skupiano się na dostarczeniu odpowiedniego oprogramowania, w jak najkrótszym czasie. Jednak bardzo szybko okazało się, że by stworzyć oprogramowanie dla klienta nie wystarczą dobre chęci i zaangażowanie programistów.

W badaniu dotyczącym analizy czynników powodzenia projektów IT [7], potwierdzone zostało, jak dużo projektów poniosło porażkę, gdy nie były ani odpowiednio zarządzane, ani nie przeprowadzały odpowiedniej analizy wymagań z klientem. Dlatego też, kluczowe w rozwoju projektów IT, było pojawienie się pod koniec lat 60 XX wieku, dziedziny nauki nazwanej inżynierią oprogramowania, zapoczątkowała ona pierwsze metody zarządzania projektami IT.

Podejść do prowadzenia projektów IT jest wiele. Jednak główny podział metod wytwarzania oprogramowania to metodyki kaskadowe i zwinne. Główną metodyką kaskadową jest dobrze znany model Waterfall, wprowadzony przez Winstona W. Royce w 1970 roku i opisany w artykule [16]. Model ten był próbą ustandaryzowania procesów i zapanowania nad chaosem produkcyjnym. Jego rdzeniem było określenie dobrego planu i konsekwentna jego realizacja. Podejście to zakładało podzielenie projektu na fazy, wzajemnie po sobie następujące (zobrazowane na rysunku 2.1). Nie można było rozpocząć kolejnej fazy wytwarzania, zanim nie skończyło się poprzedniej. Podejście było bardzo proste, schematyczne

i nastawione na dowiezienie projektu, jednak główną wadą było kosztowne wprowadzanie jakikolwiek zmian.

Rys. 2.1 Model kaskadowy [16]

W projektach IT wprowadzanie zmian jest nieuniknione, dlatego też metodyki zwinne postanowiły złamać zasady kaskadowości, nie pozwolić na zamknięte i niezależne od siebie fazy w cyklu oprogramowania i wyjść naprzeciw zmieniającym się wymaganiom klienta. O tym piszę więcej w rozdziale poświęconym metodykom zwinnym.

Wybór odpowiedniej metodyki zarządzania projektami jest kluczowy przy tworzeniu oprogramowania. Trzeba dobrać ją przede wszystkim do osób pracujących w projekcie, jak i również do rodzaju projektu. Właściwa technika ułatwi kierowanie całym procesem projektowym, wskaże drogę, jak postępować w danej sytuacji i czego należy unikać. Dlatego też wybór powinien być dokładnie przemyślany, dopasowany do klienta oraz zgodny z działalnością i polityką firmy.

3. Opis metodyk zwinnych

Jak wspomniałam we wstępie, podejściem konkurencyjnym względem kaskadowego stały się metodyki zwinne. Artykuł [9], opisuje je jako całą rodzinę metodyk trzymająca się filozofii Agile opierającej się na zasadach określonych w lutym 2001 roku w ośrodku narciarskim w Utah, przez grupę 17 praktyków. Wypracowali oni wtedy znaną nam nazwę „metodyki zwinne”, które wcześniej określane były „lekkimi”. Ponadto, udało im się określić wspominane już zasady, dokładnie określone

w Manifeście Agile, czyli inaczej manifeście zwinnego wytarzania oprogramowania. Był to swojego rodzaju protest przeciwko kaskadowemu prowadzeniu projektów. Manifest zakładał większe skupienie się na kliencie, ludziach pracujących przy projekcie oraz porzucenie zbędnej dokumentacji.

Poniżej przytoczyłam sam manifest, który dostępny jest na stronie [30]:

” W wyniku naszej pracy, zaczęliśmy bardziej cenić:

Ludzi i interakcje od procesów i narzędzi

Działające oprogramowanie od szczegółowej dokumentacji

Współpracę z klientem od negocjacji umów

Reagowanie na zmiany od realizacji założonego planu.”

Jak podkreśla sam manifest „oznacza to, że elementy wypisane po prawej są wartościowe, ale większą wartość mają dla nas te, które wypisano po lewej”. Agile daje lepsze rezultaty, bo motywuje ludzi do działania.

Podczas spotkania w 2001 roku, oprócz samego manifestu określono również zestaw 12 zasad zwinnego tworzenia oprogramowania (12 Principles) [29]. Zdecydowanie podkreślony jest w nich priorytet zadowalania klienta. Staramy się, by klient wcześnie i regularnie otrzymywał nowe wersje produktu. Nie możemy zapominać oczywiście o środowisku pracy, które powinno motywować do pracy

i być bezpieczne. Ludzie muszą czuć się pewnie w swoim otoczeniu, wiedzieć, że chcą pracować i nie boją się być innowacyjnymi. Zespół powinien cały czas skupiać się na zwiększaniu swojej wydajności. Konieczna jest dobra, ciągła komunikacja, między wszystkimi członkami zespołu. Komunikacja zapewnia transparentność w projekcie. Nacisk kładziemy tutaj na częste bezpośrednie spotkania. Nie możemy jednak zapominać, że mimo wszystko tworzymy oprogramowanie, które powinno być dobrze zaprojektowane i dążyć do technicznej doskonałości. Jedną z ważniejszych z 12 zasad, jest elastyczność wytwarzania. Błędy i zmiany będą pojawiać się zawsze, ale to od nas zależy, jak na nie zareagujemy i jak się do nich przygotujemy. W podejściu kaskadowym każda zmiana w tworzonym oprogramowaniu wiązała się z wieloma kosztami, ogromem stresu i problemów. W podejściu zwinnym błędy traktujemy jako przydatną lekcją na przyszłość i czerpiemy z nich wiele wiedzy i doświadczenia. Więcej na temat 12 zasad, można przeczytać w artykule [9].

W książce [2] przedstawione zostały największe problemy związane z wytwarzaniem oprogramowania:

- brak zaangażowania;

- niedoprecyzowane wymagania;

- nierealistyczne oczekiwania;

- zbyt mało czasu poświęconego na planowanie;

- brak reakcji na zmiany.

Analizując podane przypadki, możemy założyć, że w projektach informatycznych, nie zawodzi technologia, tylko ludzie. Każdy człowiek jest inny, każdy może mieć lepszy lub gorszy dzień. Czynnik ludzki jest niezwykle ważny. Badania opisane w artykule [25] podkreślają, że człowiek potrzebuje motywacji do działania i jego zadania muszą mieć określony cel. Oprócz samego celu, niezwykle ważna jest ocena wyników i informacja zwrotna, gdy robimy coś i nie wiemy, czy idziemy w dobrym czy złym kierunku, potrafimy być tak samo zdemotywowani, jak w momencie, gdy nasza praca nie ma celu.

W artykule [25] podkreślone jest również, jak istotna jest dla człowieka autonomia. Zespół musi czuć się bezpiecznie, że ma prawo głosu i może tworzyć zgodnie ze swoimi przekonaniami. Jest to jednak proces przekazania władzy, bardzo trudny dla kierownictwa. W programowaniu ważne jest podejmowanie szybkich i trafnych decyzji przez specjalistów, dlatego też to do nich powinny należeć decyzje związane

z wytwarzaniem oprogramowania.

Praca [24] zwraca uwagę na rolę kodu i jego uporządkowania. Podkreśla istotę ciągłej refaktoryzacji. Pomocne okazuję się również programowanie w parach. Zwiększa to efektywność wytwarzania oprogramowania, a zmniejsza błędy w kodzie. Nie można zapominać również

o retrospektywie, czyli rodzaju refleksji, gdzie powinniśmy zatrzymać się w projekcie i pomyśleć co i jak robimy i czy potrzebne są jakieś zmiany w naszym podejściu do wytwarzania oprogramowania. Agile kładzie duży nacisk na testowanie tworzonego oprogramowania. Wszystkie osoby powiązane z projektem, powinny pisać testy i testować oprogramowanie, zarówno programiści, jak i klienci. Możemy się wtedy upewnić, że system działa zgodnie z wymaganiami i zmniejszyć prawdopodobieństwo wystąpienia błędów. Przede wszystkim należy również podkreślić potrzebę ciągłego doskonalenia. Musimy zdawać sobie sprawę, że nie osiągniemy nigdy stanu idealnego, ale możemy osiągnąć stan lepszy niż jest teraz. Powinniśmy ciągle nad sobą pracować, by móc wytwarzać jak najlepsze oprogramowanie

i doskonalić proces wytwarzania.

Podejście zwinne skupia się na ludziach i interakcjach, co często mylnie uznawane jest jako podejście bez jakiejkolwiek dokumentacji. Przeciwnicy określają je jako tak zwane „robimy co chcemy”, jednak nie chodzi tutaj o to, że porzucamy całą dokumentacje i skupiamy się tylko na dostarczeniu projektu do klienta. Nacisk stawiany jest na wzajemną komunikacje i przejrzystość prac w projekcie. Kolejną ważna cechą jest zwinna adaptacja do potrzeb klienta. Zakłada się, że klient nie tyle co nie wie czego chce, co nie potrafi dobrze określić swoich potrzeb, ponieważ posiada sam zbyt małą wiedzę techniczną. Dodatkowo, świat IT pędzi do przodu, zmiany pojawiają się z dnia na dzień, klient też musi nad nimi nadążać i często wprowadzać zmiany do proejktu. Jednak właśnie o to chodzi, żebyśmy byli w stanie dostosowywać się szybko i zwinnie do wszystkich zmian i potrzeb klienta, być na zmiany gotowym

i otwartym.

Agile posiada wiele zalet tj. możliwość szybkiej reakcji na zmiany, ciągły kontakt z klientem, bieżące monitorowanie i ocena tworzonego produkty oraz zmniejszenie dokumentacji do niezbędnego minimum. Należałoby jednak również skupić się na wadach tego podejścia dokładniej opisanych

w artykule [6]. Przy dużych, schematycznych projektach, gdzie klient dokładnie wie jaki jest oczekiwany efekt końcowy, metodyki zwinne tylko komplikują wytwarzanie. W Agile zespół projektowy musi posiadać duże doświadczenie, wiele umiejętności oraz być odpowiednio zmotywowany. Sprawdza się on również raczej w mniejszych zespołach. Uwaga zespołu skupiona jest na dostarczeniu rozwiązania, nie na innych aspektach prawnych, zarządzaniu ryzykiem lub szkoleniem pracowników.

Justyna Gągała w artykule [6] przypomina, że należy również pamiętać, że metodyki zwinne nie sprawdzają się przy wszystkich projektach. Często firmy mylnie zakładają, że musza taka metodykę wprowadzić, by nadążyć za najnowszymi trendami. Jednak w momencie, gdy klient nie zgadza się na taka formę bądź wymogi prawne nie pozwalają organizacji na wprowadzenie zwinnej metodyki – nie powinno się tego wprowadzać. Agile jest głównie stosowany w realizacji projektów innowacyjnych i z dużą dawką niepewności.

Artykuł [9] przytacza wypowiedź Martina Fowlera, jednego z sygnatariuszy Manifestu, który napisał po latach: „Agile’owe metody to dla nas sposób pracy, który lubimy, a nie sedno tego, o co nam chodzi. Jeśli odkryjemy lepszy sposób pracy, zaadoptujemy go”. Chodzi o to, że jak dotąd agile’owe metody wydają się najlepszym sposobem na budowaniu użytecznego software’u. Są one mianowicie nie celem samym w sobie, a jedynie środkiem do celu.

Metodyka Scrum jest zdecydowanie najpopularniejszą metodyką zwinną, jednak nie powinniśmy zapominać, że nie jest jedyną. Firmy jednak często nie myślą o innych dostępnych rozwiązaniach, dlatego też przytoczę tutaj klika wartościowych przykładów:

3.1 Extreme programming

Extreme Programming, czyli inaczej Programowanie ekstremalne, to jedna ze zwinnych metod wytwarzania oprogramowania. Ustanowiona została pod koniec lat 90 przez Rona Jeffries’a i Kenta Beck’a. Nietypowa nazwa nawiązuje do pytania, które zadał sobie Kent „Co by się stało, gdyby realizować techniki do granic rozsądku?”. W rezultacie powstała metoda, posiadająca tylko kilka zasad, praktykowanych do ekstremum:

- ciągły przegląd kodu;

- codzienne testowanie;

- nieustająca refaktoryzacje kodu;

- krótkie iteracje.

Extreme Programming podkreśla prostotę (tworzenie, tylko tego co rzeczywiście musi zostać utworzone), odwagę (gdy to konieczne, trzeba podjąć trudne decyzje), informację zwrotną (wczesne dostarczanie produktu do klienta) i komunikację w zespole. Uważana jest również za lekki, wydajny, mało ryzykowny, elastyczny, przewidywalny, naukowy i przyjemny sposób tworzenia oprogramowania. Zmiana traktowana jest jako rzecz nieunikniona i naturalna, dlatego powinna być dobrze zaplanowana.

Krócej mówiąc cel EX to tworzenie oprogramowania o wysokiej jakości za pomocą prostych środków, przy uczciwej komunikacji wewnątrz zespołu i w relacji z klientem, ciągłej orientacji na potrzeby klienta oraz odważnym podejmowaniu trudnych decyzji.

EX opiera się na 12 praktykach, które wpasowują się w wyróżnione 4 etapy:

1. Etap planowania

Planujemy zakres pracy, tylko na najbliższą iterację. Ważne w tym etapie, jest by zespół programistyczny był świadomy, jak wybrane funkcjonalności oddziałowujących na cały system, ale również by wszystko było zrozumiałe również dla klienta jako osoby nie technicznej. Klient jest mianowicie członkiem zespołu, z którym pozostajemy w ciągłym kontakcie.

2. Etap Projektowania

Projektowanie jest analizą rozwiązania. Projekt powinien być jasny i oczywisty dla programisty, by mógł go zaimplementować, bez dodatkowych pytań. Tutaj pamiętamy, że system powinien być zaprojektowany w jak najprostszy sposób.

3. Etap Programowania

W tej metodzie, programowanie uznawane jest jako najważniejszy etap tworzenia oprogramowania. Dlatego też, trzymamy się tutaj wszystkich zasadach praktyki Agile, dodać można tylko współwłasność kodu, brak nadgodzin i praktykę programowania

w parach.

4. Etap Testowania

Pisane są testy jednostkowe oraz funkcjonalne. Dodatkowo również pisane są testy przed tworzeniem oprogramowania.

Programowanie w parach ma wiele plusów, jeśli chodzi o uzyskiwanie szybkiego i konkretnego feedbacku. Grafika 3.1 doskonale reprezentuje ten proces:

Rys 3.1 Feedback w Extreme Programming [https://commons.wikimedia.org/wiki/File%3AXP-feedback.gif]

Mariusz Chrapko w książce [2] stwierdza, że EX stanowi czasem naturalne dopełnienie metodyki Scrum, ponieważ ta nie zawiera żadnych praktyk inżynieryjnych.

Na koniec chciałabym jednak podkreślić kilka wad takiego podejścia do wytwarzania oprogramowania. Jak można zauważyć nie tworzymy dokumentacji ani specyfikacji. Wymagamy również od klienta, stałej dostępności, co często skreśla Extreme Programming na starcie. Komunikacja ustna jest bardzo pozytywna, jednak skutkuje tym, że programista jak każdy człowiek ma zawodną pamięć i gubione są informacje. Sama metoda wymaga dużego zaangażowania i samozaparcia z powodu braku procedur, struktur i dokumentów.

Szerzej na temat szczegółów metody Extreme Programming przeczytać można w książce [11] oraz artykule [12]

3.2 Lean

Lean Software Development, czyli inaczej „odchudzone” wytwarzanie oprogramowania, to kolejna zwinna metoda opierająca się tym razem na minimalizacji strat, rozumianych jako funkcje nie dodające wartości do wytwarzanego produktu. Minimalizacja ma na celu szybkie wytwarzanie

i dostarczanie oprogramowania klientowi.

Lean Development, opiera się na Lean Manufacturing, zaproponowanym w środowisku produkcyjnym Toyoty, który opierał się właśnie na eliminowaniu marnotrawstwa i ciągłym doskonaleniu produktu. Jednak za osoby, które przeniosły metodykę Lean w świat IT uważa się Mary i Toma Poppendieck. Należy pamiętać, że Lean nie określa dokładnych korków postępowania, a tylko sugeruje sposób myślenia. Opieramy się tutaj na założeniu, że doskonałość to nie moment, w którym nie mamy nic więcej do dodania, tylko moment, gdy nie mamy już nic więcej do wycięcia. Nie powinniśmy się skupiać na funkcjach, które nie są nam niezbędne, a jedynie generują dodatkową złożoność i koszty. Branża tworzenia oprogramowania jest mocno ukierunkowana na klienta, a klient jak wiadomo chce otrzymać produkty szybko i jak najtaniej.

Lean opiera się na 7 podstawowych zasadach szerzej opisanych w pozycji bibliografii [22]:

1. Eliminacja strat – jako stratę, rozumiemy wszystko co nie przyczynia się do wzrostu wartości oprogramowania.

2. Wzmaganie procesu uczenia – osiągane krótkimi cyklami przyrostowymi.

3. Opóźnianie decyzji – podejmowanie decyzji opartych na faktach i danych.

4. Jak najszybsze rezultaty – jak najszybsze dostarczanie produktu klientowi.

5. Upoważniony zespół – zespół może sam podejmować decyzje dotyczące tworzenia oprogramowania.

6. Zapewnienie spójności – zgodności z wymaganiami klient.

7. Spojrzenie całościowe – patrzymy na projekt całościowo, nie rozbijamy go na części.

Cykl wytwarzania oprogramowania w metodyce Lean przedstawiony jest na rysunku 3.2. Natomiast przegląd słabych i mocnych stron, znaleźć można w książce [11].

Rys. 3.2 Cykl wytwarzania oprogramowania w Lean [https://poradnikprzedsiebiorcy.pl/-lean-management-i-six-sigma-nowoczesne-metody-zarzadzania-firma]

3.3 Kanban

Kanban to metoda opracowana w latach 50 w Japonii. Nazwę przetłumaczyć można jako "widoczny opis" (z jęz. jap. Kan - widoczny, Ban - kartka papieru). Metoda ta opiera się przede wszystkim na przejrzystości wytwarzania. Wykorzystywana tutaj jest tablica, na której przestawiamy aktualny status wszystkich zadań. W najprostszej postaci stosowane są tylko 3 kolumny (Do zrobienia, W trakcie

i Zrobione), jednak interpretacja i ilość kolumn jest dowolna i zależna od potrzeb. Tablica pełni funkcję punktu informacyjnego i często staje się ogniwem spajającym ludzi połączonych przez wspólny cel

i proces. Opieramy się tutaj na kilku głównych zasadach, czyli wizualizacji przepływu, ograniczeniu pracy w toku, zarządzaniu przepływem, ustaleniu wyraźnych zasad i ulepszaniu.

Jako główne cele Kanbana można określić 7x żadnych:

1. żadnych braków,

2. żadnych opóźnień,

3. żadnych zapasów,

4. żadnych kolejek - gdziekolwiek i po cokolwiek,

5. żadnych bezczynności,

6. żadnych zbędnych operacji technologicznych i kontrolnych,

7. żadnych przemieszczeń.

Kanban często stosowany jest jako „dodatek” do wybranej metody wytwarzania oprogramowania, szczególnie jeśli chodzi o metodykę Scrum. Wizualizacja przebiegu pracy jest niezwykle pomocna, jednak jako osobna metoda wymaga ciągłego nadzoru i kontroli, pełnego zaangażowania pracowników. Kanban nie zakłada iteracji, tylko ciągły proces, z tego powodu sprawdza się w zespołach utrzymaniowych lub wdrażających produkty.

Przykładowa tablica Kanban przedstawiona jest na grafice 3.3.

Rys. 3.3 Tablica Kanban [http://www.everydaykanban.com/what-is-kanban/]

Szerzej w literaturze na temat metody Kanban opisuje Krystian Kaczor w książce [11] oraz Tochman w artykule [23].

4. Przegląd cech metodyki Scrum

Jednak jak wspomniałam wcześniej, zdecydowanie najpopularniejszą ze wszystkim metod zwinnych jest metodyka Scrum. Zapoczątkowana przez Kena Schwabera i Jeffa Sutherlanda w 1995 roku, poprzez opublikowanie Scrum Guide’a [18], czyli zbioru zasad opisujących metodykę Scrum. Jednak są to tylko opisane ramy postępowania, implementacja jest otwarta i możliwe jest stosowanie różnych procesów i technik. Jedną z idei metodyki Scrum, jest ciągłe doskonalenie produktu, zespołu oraz środowiska pracy.

Metodykę Scrum porównać można do gry w piłkę nożną. Każdy dobrze zna zasady, są one dokładnie określone, jednak nie każda drużyna wygrywa. Tak samo jest w przedstawianej metodyce, zasady zna każdy, ale implementacja jest zależna od strategii. Jak podkreśla Charpko w książce [2] „Scrum jest szkieletem, a nie w pełni rozwiniętym procesem. Implementacja pozostawia dużo przestrzeni”. Metodyka Scrum nie przedstawia procesu nakazowego, nie opisuje co należy robić w każdej sytuacji, oferuje jedynie szkielet zachowań, które utrzymują proces wytwarzania oprogramowania na widoku.

Nazwa Scrum jest terminem zaczerpniętym z rugby, oznacza to „Młyn”. Jest to pozycja graczy (charakterystyczny młyn/koło) przyjmowana w celu rozpoczęcia gry po przerwie.

Metodyka Scrum przenosi kontrole z zarządu na indywidualne zespoły, poprzez przekształcanie zespołów w zarządców zdanych na samych siebie. Takie podejście znacznie skraca czas sprzężenia zwrotnego z klientem, co jest istotne, gdy rynek nie jest stabilny i wymagania szybko się zmieniają.

W metodyce Scrum podejmujemy decyzje w oparciu o fakty i zebrane doświadczenie. Skupiamy się również na dostarczeniu najwyższych priorytetów dotyczących rozwiązania. Metodyka Scrum uwydatnia nam nasze błędy, pokazuje, gdzie co dzieję się źle i gdzie możemy się dostosować. Istotą Metodyki jest zmiana.

Metodyka Scrum skupia się na ciągłej nauce i doskonaleniu, tworzy uczące się środowisko pracy. Posiadanie dokładnej instrukcji postępowania krok po kroku w metodyce Scrum byłoby zbyt restrykcyjne. Metodyka ta zapewnia bezpieczne otoczenie, gdzie ludzie przyznają się do porażek I nie boją prosić

o pomoc. Dodatkowo zachęca do różnych rodzajów nauki, ponieważ każdy uczy się inaczej. Opiera się na wizualizacji informacji, przez co są bardziej otwarte dla wszystkich. Zapewnia jeden wspólny cel, do którego dążymy.

Jak wspomniane jest w samym Scrum Guide – metodyka Scrum jest lekka i łatwa do zrozumienia, ale trudna do opanowania. Jednakże jak zauważa Ken w książce [19] „Scrum jest trudny, jednak nie dzieje się tak z powodu rzeczy, które robimy, ale z powodu rzeczy, których nie robimy”.

Metodyka Scrum nie powinna być wykorzystywana w chaotycznych projektach, gdzie wymagana jest szybka reakcja i trzeba działać natychmiastowo. Gorzej radzi sobie również w pacy sterowanej przerwaniami – pracy utrzymaniowej. W prostych, dobrze nam znanych projektach można użyć omawianej metodyki, ale jeśli dokładnie wiemy czego chcemy i jak będzie wyglądał projekt, Metodyka Scrum może tylko zamieszać. W tej metodyce nie jesteśmy w stanie ocenić ram projektu, skupia się ona bowiem na iteracyjnym przyroście oprogramowania. Natomiast metodyka Scrum bardzo dobrze sprawdza się w projektach pilnych i w tych, gdzie wymagania są nieznane lub ciągle zmieniające się.

Książkami poszerzającymi temat metodyki Scrum, opisującymi dokładnie wady, zalety i propozycję wdrożenia będą pozycje bibliografii [2,17,19]. Artykuł opisujący, dlaczego metodyka Scrum działa i jak powinno się z niej korzystać [3].

4.1 Główne zasady

Metodyka Scrum opiera się na teorii empirycznej, twierdzącej, że wiedza wynika z doświadczenia

i podejmowania decyzji w oparciu o zdobytą wiedzę. Empiryzm ugruntowany jest na 3 filarach:

1. Przejrzystości – wszystkie aspekty istotne w projekcie powinny być widoczne i klarowne dla wszystkich uczestników procesu.

2. Inspekcji – ciągłego kontrolowania i sprawdzania procesu w celu odkrycia rozbieżności i działań niepożądanych.

3. Adaptacji –dostosowywaniu się do zmian i korygowaniu błędów.

Niezwykle ważne są natomiast wartości metodyki Scrum. W artykule [13] Autor bardzo trafnie przyrównuje metodykę do góry lodowej. To co widzimy ponad wodą to ten czubek – czyli role, artefakty, zdarzenia. Łatwo je oczywiście zaimplementować i się ich trzymać, jednak to co tworzy metodykę scrum to wartości za nią stojące, czyli:

1. Zaangażowanie

2. Odwaga

3. Skupienie

4. Otwartość

5. Poszanowanie

Wszyscy członkowie zespołu scrumowego zobowiązani są postępować zgodnie z wartościami. W ten sposób tworzona jest atmosfera zaufania.

4.2 Role

Zespół scrumowy jest przede wszystkim samoorganizujący się. Oznacza to, że zespół nie ma żadnego zarządcy spoza zespołu i sam rozdziela zadania i obowiązki. Jest również międzyfunkcjonalny, co oznacza, że sam jest w stanie dostarczyć rozwiązanie bez pomocy innych zespołów. W zespole scrumowym określone są 3 role:

4.2.1 Scrum Master

Scrum Master odpowiedzialny jest za promowanie i przestrzeganie zasad metodyki Scrum. Jego priorytetem jest dbanie o dobro i ciągłość pracy zespołu. Scrum Master wspiera zespół programistyczny szkoląc ich w zakresie samoorganizacji i międzyfunkcjonalności, pomaga w tworzeniu produktu usuwając przeszkody i w przeprowadzaniu wydarzeń scrumowych. Służy on pomocą Product Ownerowi, zapewniając, że cele, zakres i domena produktowa są zrozumiałe dla zespołu. Scrum Master jest odpowiedzialny za szukanie nowych technik do adaptacji, zrozumienia i praktykowania zwinności. Wspiera on również organizacje przewodząc procesom wdrażania metodyki Scrum, planując wykorzystanie metodyki Scrum w organizacji oraz wspierając pracowników i interesariuszy w jej stosowaniu.

Scrum Master jest w pewien sposób odpowiednikiem Kierownika Projektu w kaskadowym podejściu do zarządzania projektem. Jednak nie możemy tych ról przyrównywać. Scrum Master bowiem nie posiada władzy nad zespołem, tylko władzę nad procesem. Powinien on proponować rozwiązania zespołowi i kierować ich na jak najlepszą ścieżkę, jednak ostateczna decyzja zawsze należy do zespołu. Jego rola to tworzenie okazji dla zespołu – by pracowali lepiej, tworzyli zgrany zespół i pomagali sobie nawzajem.

Według książek [2, 17] Scrum Master powinien odznaczać się określonymi cechami:

- odważny;

- nie żądny krwi;

- chętny do współpracy;

- obserwator;

- odpowiedzialny;

- wpływowy;

- podsiada szeroką wiedzę z metodyki Scrum;

- cierpliwy;

- proaktywny.

4.2.2 Product Owner

Jest to osoba (jedna!) odpowiedzialna za maksymalizacje wartości produktu i pracy zespołu developerskiego. Product Owner to osoba reprezentująca interesy klienta. Tak jak dla Scrum Mastera priorytetem było dobro zespołu, tutaj priorytetem jest zadowolony klient. Jest to bardzo dobre podejście, ponieważ istnieją dwie role, które dbają o dobro dwóch stron, co równoważy proces.

Product Owner jest osobą, która reprezentuje interesy każdej z osób zainteresowanych projektem i docelowym systemem. Jest on przede wszystkim odpowiedzialny za Backlog Produktu. Tworzy go, poprawia, uporządkowuje, zapewnia jego przejrzystość, określa cele i misje, optymalizuje wartość pracy wykonanej przez zespół developerski. Product Owner może zlecać poszczególne z tych zadań zespołowi developerskiemu, jednak w dalszym ciągu pozostaje on za nie odpowiedzialny.

Według książek [2, 17] Product Owner powinien odznaczać się określonymi cechami:

- wizjoner;

- inspirator;

- decyzyjny;

- dostępny i zaangażowany;

- negocjator;

- silne umiejętności interpersonalne;

- mający władze;

- posiada ekspercką wiedzę biznesową i domenową;

- konkretny;

- odpowiedzialny.

4.2.3 Zespół Developerski

Istotą metodyki Scrum jest niewielki zespół, od 3 do 9 członków. Zbyt mały zespół oznacza mniejszy wzrost w każdej iteracji, a zbyt duży wymaga sporych nakładów pracy idących w koordynacje. Zespół developerski składa się z profesjonalistów, nie określanych rolami, tylko posiadających zestawy umiejętności. Nie wyróżniamy również podzespołów (testujących, analizujących itd.). Tylko zespół developerski odpowiada za utworzenie przyrostu produktu, dlatego samodzielnie organizuje własną pracę. Odpowiedzialność za całą pracę ponosi cały zespół. Zespół powinien być grupą ludzi pracującą razem dla osiągnięcia wspólnego celu.

Według książek [2, 17] zespół developerski powinien odznaczać się określonymi cechami:

- zgrany;

- samoorganizujący;

- samowystarczalny/mulitidyscyplinarny;

- posiadający umiejętności typu T (szeroka wiedza specjalistyczna);

- odpowiedzialność zbiorowa – wspólnota kodu;

- skupiony i zaangażowany.

4.3 Wydarzenia

Wszystkie spotkania określone w Scrum Guide to punkty przeprowadzania inspekcji i możliwości zastosowania adaptacji. Dodatkowo wydarzenia zapewniają przejrzystość w projekcie. Nie powinno się wprowadzać dodatkowych cyklicznych spotkań. Wszystkie spotkania są ścisłe ograniczone czasowo. Należy również pamiętać, że pominięcie któregokolwiek z wydarzeń, może skutkować obniżeniem przejrzystości w projekcie i utraceniem możliwości inspekcji i adaptacji.

4.3.1 Sprint

Sprint to czas jednej iteracji o ściśle określonym czasie trwania. Nie powinien on być ani skracany, ani wydłużany, ponieważ zaburza to predykcję i planowanie. Sprint może trwać od tygodnia do miesiąca.

Zespół po kilku sprintach jest w stanie określać i planować, ile czasu zajmie mu wdrożenie ilu zadań, zaburzanie czasu sprintu powoduje tego zachwianie.

Na ilustracji 4.1 przedstawiony jest przebieg trwania jednego sprintu.

Rys. 4.1 Sprint w metodyce Scrum [https://www.qagile.pl/scrum/scrum-opis/]

Każdy sprint powinien rozpoczynać się planowaniem, a kończyć przeglądem i retrospektywą.

W każdy dzień trwania sprintu powinno być przeprowadzane daily (spotkanie codzienne). Należy pamiętać, że każdy sprint musi kończyć się wytworzeniem zamkniętej i skończonej funkcjonalności. Inaczej mówiąc, każdy sprint powinniśmy traktować jako osobny projekt, który musi zostać ukończony i tworzyć spójny przyrost.

Dodam, że możliwe jest przerwanie trwania sprintu, jednakże nie jest to zjawisko częste, ani tym bardziej pożądane. Prawo do przerwania sprintu leży w rękach tylko i wyłącznie Product Ownera. Taką decyzję powinien podjąć tylko jeśli Cel Sprintu zostanie zdezaktualizowany, w skutek zmian strategii lub rynku. Powinien ją również podjąć po dokładnych dyskusjach z resztą zespołu i interesariuszami.

4.3.2 Planowanie sprintu

Planowanie sprintu jest spotkaniem rozpoczynającym dany sprint, podczas niego obecni są wszyscy członkowie zespołu scrumowego i ustalana jest praca na najbliższy sprint. Spotkanie powinno trwać maksymalnie 8 godzin, w momencie, gdy sprint trwa miesiąc. Scrum master powinien zapewnić przejrzystość spotkania, możliwość jego odbycia się i zapewnienie, że wszyscy rozumieją jego cel. Natomiast Właściciel Produktu uczestniczy w spotkaniu w celu rozwiania jakikolwiek niepewności

i wytłumaczenia wszystkich elementów Backlogu.

Podczas spotkania określane jest co zostanie dostarczone w ramach sprintu i jak zostanie wykonana praca niezbędna do jego dostarczenia. Zespół developerski sam decyduje, ile elementów Backlogu jest w stanie dowieść. Tworzą w ten sposób Backlog Sprintu. Podczas tego spotkania określany jest również Cel Sprintu, pozwala to na podniesienie przejrzystości dla zespołu developerskiego. Na to spotkanie mogą być również zaproszone inne osoby spoza zespołu, wedle potrzeb.

Opis spotkania i pytań, które warto na nim zadać oraz wszystkie reguły i ograniczenia dostępne są w książce [19].

Na koniec należy pamiętać, że jak podkreśla Mariusz Chrapko w książce [2] „Musimy pamiętać, że nie jesteśmy w stanie przewidzieć wszystkiego z góry, planowanie powinno być pomocne, ale nie przesadne”.

4.3.3 Przegląd sprintu

Przegląd sprintu jest spotkaniem kończącym okres jednego sprintu. Podczas tego spotkania inspekcji poddawana jest wykonana praca, czyli przyrost produktu. Oceniane jest co udało się zrobić,

a czego się nie udało. Omawiane są napotkane podczas sprintu przeszkody i ich rozwiązania. Na tej podstawie powinny być wyciągane wnioski, co do planowania pracy na kolejne sprinty. Analizowane jest również, jakie kroki powinny zostać poczynione, by zoptymalizować kolejne przyrosty. Spotkanie ma przede wszystkim na celu pobudzenie współpracy oraz uzyskanie informacji zwrotnej. Wynikiem przeprowadzonego spotkania jest poprawiony Backlog.

Spotkanie to trwa maksymalnie 4 godziny dla miesięcznego sprintu. Scrum Master odpowiedzialny jest za zapewnienie, że spotkanie się odbywa i jego uczestnicy rozumieją cel. Podczas spotkania powinni być obecni interesariusze. Spotkanie ma na celu usprawnienie pracy i uczynienie jest bardziej satysfakcjonującą. Oczywiście należy pamiętać, że wszelkie ustalenia podczas spotkania musza być zgodne ze strategia firmy i produktu.

Przy pomyślnie przeprowadzonym przeglądzie, ludzie niebędący członkami zespołu scrumowego dowiadują się o przeprowadzonych pracach i pomagają wyznaczyć ich dalszy kierunek. Zespół scrumowy, ma natomiast szanse podnieść wartość biznesowa i marketingowa na podstawie informacji zwrotnej.

4.3.4 Retrospektywa

Tak jak przegląd sprintu dotyczy inspekcji i adaptacji produktu, tak retrospektywa dotyczy inspekcji i adaptacji PROCESU. Celem retrospekcji jest nieustanna poprawa procesu, dzięki której dobry zespół scrumowy, staje się jeszcze lepszym zespołem scrumowym.

Retrospektywa sprintu przeprowadzana jest zazwyczaj zaraz po spotkaniu dotyczącym przeglądu sprintu. Podczas tego spotkania, zespół nie skupia się na przyroście, ale na wzajemnej współpracy. Zespół wspólnie zastanawia się nad tym co zrobić, by usprawnić współprace oraz prace nad produktem. Brane pod uwagę są różne aspekty ludzi, relacji, procesów oraz narzędzi. Spotkanie nie powinno trwać dłużej niż

3 godziny dla sprintu o miesięcznej długości.

Scrum master, oprócz typowych obowiązkowych dotyczących spotkań, zapewnia również, że spotkanie przebiega w dobrej atmosferze i jest produktywne.

4.3.5 Daily

Daily, czyli inaczej codzienny scrum to spotkanie, które powinno być przeprowadzane codziennie i nie trwać dłużej niż 15 minut. Jest to wydarzenie skierowane głównie do zespołu developerskiego, żeby mogli podzielić się postępem w projekcie. Podczas spotkania każdy z członków zespołu musi się wypowiedzieć i opowiedzieć co udało mu się zrobić wczoraj i czym zajmować będzie się dzisiaj.

Jest to sposób na kontrole i motywacje do działania dla członków zespołu, jak i również miejsce, gdzie zapewniamy komunikacje i przejrzystość w projekcie. Jest to możliwość zaplanowania czasu na kolejne 24 godziny. Spotkanie powinno odbywać się codziennie w tym samym miejscu o tej samej porze. Oceniany jest na nim postęp prac i weryfikacja celu sprintu. Podczas spotkania członkowie alarmują również o wszystkich napotkanych oraz potencjalnych przeszkodach. Pozwala to na jak najszybsze zidentyfikowanie problemów i ich naprawę. Dodatkowo daily pozwala na przyspieszenie procesu podejmowania decyzji. Spotkanie to jest niezwykle kluczowe, jeśli chodzi o aspekty inspekcji i adaptacji.

Scrum Master zapewnia, że spotkanie na pewno się odbędzie i odbędzie się w wyznaczonych ramach czasowych, jednak to zespół developerski jest odpowiedzialny za jego przeprowadzenie.

4.4 Artefakty

4.4.1 Backlog

Backlog Produktu, to lista wszystkich rzeczy, które na ten moment wiemy o rozwoju produktu. Jest to jedyne źródło wymagań w projekcie. Osobą odpowiedzialną za jego tworzenie i pielęgnowanie jest Właściciel Produktu. Nie jest to lista zadań, która w którymś momencie zostanie ukończona, Backlog rozwija się cały czas – dodajemy nowe elementy, usuwamy te które się przeterminowały, uszczegóławiamy te zbyt duże. Backlog istnieje tak długo, jak istnieje tworzony na jego podstawie produkt.

Backlog produktu powinno się regularnie doskonalić, dodawać szczegóły, szacowania, porządkować elementy. Backlog powinien posiadać opisane Definition of Done, czyli definicje ukończenia. Jest to zdefiniowany zestaw kroków, które element musi przejść, by uznano go za zakończonego. Przykład listy można znaleźć znaleźć w książce [2].

Rys. 4.2 Backlog projektu [https://resources.collab.net/blogs/product-backlog-is-deep-invest-wisely-and-dive-carefully]

Dobry rejestr produktu cechuje się DEEP, czyli szczegółowy, oszacowany, przygotowany na zmiany i zpriorytetyzowany. Zadania o najwyższym priorytecie, powinny znajdować się na górze listy. Również te zadania powinny opisane być zgodnie z zasada INVEST – czyli wartościowe, odrębne, wyestymowane, odpowiedniej wielkości, określone i testowalne.

4.4.2 Backlog Sprintu

Backlog sprintu, to zbiór wszystkich zadań wybranych podczas spotkania planowania sprintu, do wykonanie w najbliższym sprincie. Obrazuje on całą prace, jaką należy wykonać, by osiągnąć cel sprintu. Backlog Sprintu, jest modyfikowany w trakcie trwania sprintu, gdy developerzy zaczną pracę nad zadaniami, często wyłania się więcej szczegółów. Jednak to tylko developerzy mogą go zmieniać. Jest to wyłączna własność zespołu developerskiego.

4.5 Scrum of Scrums

Często spotykaną praktyką, przy dużych projektach jest scrum scrumów. W momencie, kiedy trzeba podzielić prace nad jednym produktem na wiele zespołów, scrum scrumów pozwala na koordynowanie swojej pracy wewnątrz zespołu, mają na uwadze pracę innych zespołów. Zespół dodatkowo są w nim osoby, które są przedstawicielami zespołu dla innych zespołów developerskich. Organizowane jest wtedy dodatkowe spotkanie, które na celu ma synchronizacje prac wszystkich zespołów.

Więcej informacji jak przeprowadzać scrum of scrum, o odpowiednich narzędziach i pułapkach znaleźć można w artykule [1].

Metodyka Scrum jest jedną z najbardziej popularnych metodyk zwinnych, implementowanych jednak bardzo często „po swojemu”. Eliminując niezbędne elementy lub dodając wykraczające poza zakres opisany w Scrum Guide.

W metodyce Scrum niezwykle ważne jest to, by zespół i osoby związane z projektem rozumiały po co jest on wprowadzany. Istotne okazuje się również przygotowanie firmy i klienta na wprowadzenie metodyki Scrum do projektu.

5. Analiza adaptacji metodyki Scrum w firmach

W Scrum Guide [18] napisane jest: „Poszczególne zastosowania metodyki Scrum mogą się między sobą różnić i z tego względu wykraczają poza zakres tego przewodnika”.

Dlatego też, zaprojektowałam i przeprowadziłam badanie mające na celu zbadanie różnorodności we wdrażaniu metodyki Scrum w zespołach wytwórczych. Przygotowałam ankietę składającą się z 23 pytań zamkniętych i 18 pytań otwartych. Pytania dotyczyły adaptacji metodyki Scrum w różnych firmach, w różnych projektach. Pytania podzieliłam na 3 sekcje przedstawione w tabeli 5.1. Celem pierwszej sekcji, było określenie które artefakty i role opisane w Scrum Guide miał dany zespół. Druga sekcja dotyczyła natomiast implementacji i zastosowań reguł scrumowych. Natomiast ostatnia trzecia sekcja miała na celu zbadanie subiektywnej oceny ankietowanego na temat metodyki Scrum.

Ankieta została rozpowszechniona na popularnych portalach społecznościowych (tj. Facebook, LinkedIn). Udostępniona została na grupach tematycznych, do których otrzymałam dostęp, zrzeszających osoby powiązane z Agile i metodyką Scrum. Przeprowadziłam również 20 wywiadów bezpośrednich,

w celu uzyskania dokładniejszych wyników. Otrzymałam 100 odpowiedzi i uznałam tą grupę za grupę reprezentatywną, pośród osób znajdują się studenci od początkowych do końcowych lat studiów, młodzi absolwenci oraz osoby pracujące w branży od kilku do kilkunastu lat. Wywiady były przeprowadzane zarówno z programistami, jak i Scrum Masterami i Product Ownerami.

Sekcja Nr. Pytanie

Sekcja 1 1.1 Czy w zespole jest Scrum Master?

1.2 Czy w zespole jest Product Owner?

1.3 Czy jest stworzony Backlog projektu?

1.4 Czy w projekcie stosowany jest podział na sprinty?

1.5 Czy przeprowadzane jest spotkanie dotyczące planowania sprintu?

1.6 Czy przeprowadzane jest spotkanie dotyczące przeglądu sprintu? (note: przegląd sprintu to przegląd wykonanej pracy - co zostało dowiezione, a co nie)

1.7 Czy przeprowadzane jest spotkanie dotyczące retrospekcji sprintu? (note: retrospektywa sprintu to analiza wspólnej pracy - jak nam się pracowało, co możemy poprawić, by pracowało się lepiej?)

1.8 Czy przeprowadzane są daily (spotkania codzienne)?

1.9 Czy korzystacie z tablicy KANBAN? (na przykład za pomocą programu JIRA)

Sekcja 2 2.1 Czy czas trwania sprintu jest stały? Jeśli tak, to ile trwa?

2.2 Czy tworzony jest Backlog sprintu?

2.3 Czy każdy sprint ma określony cel?

2.4 Czy podczas sprintu pojawiają się "wrzutki" zadań?

2.5 Czy każdy sprint dostarcza ukończoną funkcjonalność?

2.6 Kto z zespołu scrumowego uczestniczy, w których spotkaniach? (note: Jeśli uczestniczą wszyscy - zaznacz kolumnę wszyscy) - tutaj zastosowałam macierz do zaznaczania kto uczestniczył

2.7 Ile trwa spotkanie dotyczące planowanie sprintu?

2.8 Ile trwa spotkanie dotyczące przeglądu sprintu?

2.9 Ile trwa spotkanie dotyczące retrospekcji sprintu?

2.10 Ile trwa daily?

2.11 Kto ustala Backlog projektu?

2.12 Czy backlog jest często zmieniany? Jeśli tak, to kto go zmienia?

2.13 Czy do opisu elementów Backlogu używane są User Stories? (note: Jako ... chciałabym ..., w celu…)

2.14 Czy każdy Backlog ma określone Definition of Done?

2.15 Czy używana jest metoda MoSCoW? (grupowanie Backlogu pod kątem priorytetu: Must, Should, Could, Want)

2.16 Kto przydziela zadania poszczególnym członkom zespołu? W jaki sposób rozdzielana jest praca?

2.17 Czy nadawane są priorytety do zadań, jeśli tak to kto je nadaje?

2.18 Czy ustalana jest czasochłonność zadań, jeśli tak to kto ją ustala?

2.19 Ile osób liczy zespół programistyczny?

2.20 Czy developerzy pracują przy jednym projekcie czy wielu?

2.21 Czy developerzy są pełnoetatowi?

2.22 Czy developerzy w zespole posiadają stricte określone role? Czy raczej każdy z posiada swój "zestaw umiejętności"?

2.23 Czy zespół pracuje sam nad dostarczeniem sprintu, czy potrzebuje developerów z innych zespołów? (np. testerzy, administratorzy)

2.24 Czy tworzony jest wykres wypalania?

2.25 Czy Scrum Master i Product Owner to ta sama osoba?

2.26 Czy istnieją inne role oprócz wspomnianych? (np. Project Manager, Technical Lead)

2.27 Czy istnieją inne spotkania oprócz wspomnianych? (takie, które są cyklicznie ustawione w kalendarzu)

2.28 Jakich narzędzi używacie do wspierania metodyki Scrum - szczególnie jeśli chodzi o komunikację? (np. JIRA, Enterprise Architect, Skype, Slack)

Sekcja 3 3.1 Czy odnosisz wrażenie, że spotkanie codzienne Ci pomaga? (chodzi o twoje odczucia, czy to spotkanie jest i każdy musi być, czy pomaga przy zrozumieniu statusu projektu)

3.2 Jakbyś określił role i pracę Scrum Mastera? Czy rzeczywiście "usuwa przeszkody"? Czy pilnuje, żeby metodyka Acrum była przestrzegana? (Jeśli nie masz Scrum mastera w zespole, pomiń to pytanie)

3.3 Co sądzisz o metodyce Scrum w swoim projekcie?

Tabela 5.1 Pytania do ankiety

Wyniki ankiety przestawiłam na wykresach od 5.1 do 5.50. W tabeli 5.2 przedstawiony został rozkład branż firm, których projekty pracujące w metodyce Scrum, opisywane były w ankiecie. Jak można zauważyć, wyniki są bardzo różnorodne, jednak wiele z nich dotyczyło firm sprzedających oprogramowanie i firm medycznych lub farmaceutycznych.

Branża Liczba

IT 14

Medyczna/ Farmaceutyczna 11

software house 11

Brak Odpowiedzi 10

bankowość 6

Transport 5

Telekomunikacja 4

Ubezpieczenia 4

e-commerce 3

Gry 3

Big data 2

Finanse 2

urządzenia ADG i mobilne 2

Wspieranie działań w obszarach HR i komunikacji 2

Administracja publiczna 1

Automatyka 1

benefit system 1

Mar-tech 1

Business Inteligence 1

Dom maklerski 1

Energetyka 1

Fmcg 1

Information Security 1

Consulting 1

Marketing 1

Media 1

Muzyczna 1

Oil&Gas 1

Perform 1

Podatki 1

Podróże 1

Robotyka 1

Shared service 1

Wsparcie edukacji 1

Wydawnictwo + serwisy online 1

Tabela 5.2. Branże

Pierwsze pytanie 1.1 dotyczyło istnienia roli Scrum mastera w zespole. Jak widzimy większość zespołów miała Scrum Mastera, jednak aż 24% projektów działały bez tego niezbędnego członka zespołu scrumowego. W odpowiedziach „Inne” zawierają się zespoły ze Scrum Masterem, ale gdzie inna osoba obejmowała tą rolę tj. Technical Lead, Team Lead, jeden z programistów, Scrum Coach, Project Manager

i Team Leader nad zespołem i Project Manager nad projektem.

Rys. 5.1. Scrum Master w zespole

Pytanie 1.2 dotyczyło tego samego zagadnienia, jednak tutaj pytanie wiązało się z rolą Product Ownera. Na Rys. 5.2. przedstawiony jest rozkład odpowiedzi. Widzimy, że z tej roli zespoły rezygnują rzadziej, niż ze Scrum Mastera, jednak również tutaj, aż 16% zespołów pracowało bez Product Ownera. Co ciekawe, w tych liczbach zawiera się 9% zespołów, w których nie istniała rola ani Scrum Mastera, ani Product Ownera. W odpowiedzi „Inne” zawierają się teraz sytuacje, gdzie każdy klient miał osobnego Product Ownera lub zamiast Product Ownera, było kilka osób od klienta. W niektórych projektach rolę Product Ownera przejmowali również analityk, Key Account Manager i PM.

Rys. 5.2 Product Owner w zespole

Pytanie 1.3 dotyczące posiadania Backlogu w projekcie, wypadło bardzo dobrze. Tylko 3% projektów nie posiadały Backlogu co widzimy na Rys 5.3. Niemniej jednak, jest to zastanawiające, w jaki sposób określali wymagania do projektu. Dodatkowo 2% z tych projektów nie istniała również roli Scrum Mastera i Product Ownera w zespole. Sytuacja ta dotyczyła branży transportowej i Shared Service.

Rys. 5.3 Backlog w projekcie

Pytanie 1.4 dotyczyło stosowania podziału na sprinty, gdzie wyniki przedstawione są na Rys. 5.4. W tym przypadku, tylko 2% projektów posiadało podziału na sprinty i były to 2 firmy, o których mowa była przy pytaniu numer 3 (nie posiadały Backlogu i nie istniała rola Scrum Mastera i Product Ownera

w projekcie).

Rys. 5.4 Sprinty w projekcie

Pytanie 1.5 dotyczyło przeprowadzania spotkania planującego sprint, wyniki zostały przedstawione na Rys. 5.5. Tylko 2% projektów rezygnowało z tego spotkania i były to projekty bez Scrum Mastera i Product Ownera, więc zapewne bez osób będących w stanie pomóc z przeprowadzeniem spotkania.

Rys. 5.5 Spotkanie planowania sprintu

Pytanie 1.6 dotyczyło przeprowadzania spotkania przeglądu sprintu. Wyniki zobrazowane zostały na Rys. 5.6. Jak można zauważyć większość projektów przeprowadzała to spotkanie, 6% zespołów zdecydowało się na łączenie go z planowaniem, a 6% zespołów nie przeprowadzało go w ogóle. Pod hasłem „Inne” opisane są sytuacje, gdzie przegląd nie był przeprowadzany po każdym sprint’cie, demo było przeprowadzane na bieżąco lub gdy przegląd sprintu nie był przeprowadzany od razu w trakcie działalności zespołu.

Rys. 5.6 Spotkanie przeglądu sprintu

Pytanie 1.7 dotyczyło retrospektywy, a wyniki zostały przedstawione na Rys. 5.7. Jak widać 4% zespołów zdecydowało się na łączenie tego spotkania z planowaniem i również 4% połączyły go

z przeglądem. Natomiast, aż 14% zespołów rezygnowało z jego przeprowadzania. Pod odpowiedziami „Inne” kryje się retrospekcja po stronie dostawcy z zespołem programistycznym, bez Product Ownera oraz odpowiedź, że zazwyczaj jest jednak członkowie zespołu nie są chętni uczestniczyć w tym spotkaniu.

Rys. 5.7 Retrospektywa sprintu

Pytanie 1.8 dotyczyło przeprowadzania spotkania codziennego (daily), a wyniki przedstawione zostały na Rys. 5.8. Jak widzimy tylko 5% zespołów rezygnowało z tego spotkania. Pod „inne” możemy znaleźć odpowiedzi tj. daily przeprowadzane co dwa dni, niecodziennie oraz odpowiedź, że czasem się odbywa, jeśli nie zostanie zapomniane, bo większość zespołu nie widzi w nim korzyści.

5.8 Spotkanie codzienne

Ostatnim pytaniem z 1 sekcji ankiety było pytanie 1.9 dotyczące kozystania z tablicy Kanban,

a wyniki zostały zobrazowane na Rys. 5.9. Jak widać aż 88% zespołów decydowało się na użycie tablicy, gdzie nie jest ona obowiązkową wytyczną metodyki Scrum, jedynie zaleceniem. Odpowiedź „Inne” dotyczyła sytuacji, gdzie tablica była, jednak wszystkie zadania znajdowały się w kolumnie „in progress”.

Rys. 5.9 Tablica Kanban

Pierwsze pytanie z sekcji drugiej 2.1 odnosiło się do tego, czy czas trwania sprintu jest stały i jeśli tak to, ile trwa. Dlatego też, podzieliłam to pytanie na 2 części, wyniki pierwszej zobrazowałam na Rys 5.10, następnie na Rys 5.11 przedstawiony został czas trwania sprintu w poszczególnych projektach, gdzie czas trwania sprintu był stały. Jak można zauważyć 83% zespołów trzymała się ustalonego czasu trwania. W odpowiedziach „nie” można było znaleźć odpowiedzi takie jak 1-2 tygodnie, 2-4 tygodnie, na początku miesiąc jednak zostało zmienione na 2 tygodnie oraz sytuacja, gdzie 1 sprint równał się 1 dużej zmianie. Patrząc na czas trwania stałych sprintów, zauważyć można dużą przewagę sprintów trwających 2 tygodnie lub 1 tydzień.

Rys. 5.10 Stały czas sprintu w projekcie

Rys. 5.11 Czas trwania sprintu

Pytanie 2.2 dotyczyło tworzenia Backlogu sprintu, a wyniki przedstawione są na Rys. 5.12. Jak widać, zdecydowana większość zespołów (92%) decydowała się na dokładne określenie Backlogu sprintu. Siedem zespołów rezygnowało z tego podejścia, z czego aż 4% z tych zespołów nie miały Scrum Mastera, a 3% również Product Ownera.

Rys. 5.12 Backlog Sprintu

Pytanie 2.3 dotyczyło określania celu sprintu i analizując wyniki na Rys. 5.13 widzimy jak wiele zespołów nie decydowało się na określanie celu. W odpowiedziach „inne” zawarte były stwierdzenia określania celu na 1-2 miesiące lub na 2 sprinty. Korzystanie z metody nadawania celów sprintu, niesie za sobą wiele korzyści. Określony cel nie tylko wprowadza przejrzystość i transparentność celu biznesowego na dany sprint, ale też wspomaga komunikację z interesariuszami, zwięźle komunikując ich o aktualnie przebiegających pracach nad produktem.

Rys. 5.13 Cel sprintu

Pytanie 2.4 dotyczyło „wrzutek” zadań, podczas trwania sprintu, czyli naruszania Backlogu Sprintu. Wyniki zobrazowane na Rys. 5.14 wyraźnie pokazują, że aż 62% zespołów borykała się z problemem wpadających zadań. Pod „inne” kryją się odpowiedzi takie jak czasem, rzadko i propozycja dodania do Backlogu sprintu zadania obejmującego obsługę bugów. Często dodawane nowe zadania do zaplanowanego sprintu, obciążają sprint i zakłócają pracę programistów. Powoduje to tworzenie stresującej atmosfer, nadgodzin w projekcie lub problemy z dostarczaniem ustalonych na planowaniu zadań.

Rys. 5.14 Wrzutki zadań

Pytanie 2.5 dotyczyło dostarczania ukończonej funkcjonalności na koniec każdego sprintu. Na Rys. 5.15 widzimy, że prawie połowa zespołów nie była w stanie dostarczać ukończonego rozwiązania. Tylko niewiele ponad 1/3 zespołów dostarczało ukończone rozwiązanie. Jest to duży problem po stronie biznesu i klienta, zaakceptowanie metodyki Scrum, gdy widzimy, że nie dostarcza tego co potrzebujemy i na co się „umawialiśmy” podczas planowania sprintu.

Rys. 5.15 Ukończona funkcjonalność pod koniec sprintu

Następna seria dotycząca pytania 2.6 dotyczyła obecności członków zespołu scrumowego na poszczególnych spotkaniach. Na Rys. 5.16 widzimy wykres dotyczący spotkania planowania sprintu.

W zdecydowanej większości uczestniczą w nim wszyscy członkowie zespołu, wiele odpowiedzi, gdzie nie byli obecni jacyś członkowie odnosiła się do sytuacji, gdzie w zespole nie było Scrum Mastera lub Product Ownera. Rys. 5.17 przedstawia wykres dotyczący spotkania przeglądu sprintu. Rozkład tutaj jest zbliżony do rozkładu dla spotkania dotyczącego planowania. Kolejny wykres na Rys. 5.18 przedstawia rozkład dla spotkania dotyczącego retrospektywy sprintu. Tutaj rozkład się zmienia w stosunku do wcześniejszych wykresów, widzimy zmianę dotyczącą obecności Product Ownera, która się zmniejsza na rzecz obecności Scrum Mastera. Na koniec przyjrzyjmy się Rys. 5.19, który przedstawia rozkład dla daily. W tym wypadku, zmniejsza się liczba w kolumnie „wszyscy” na rzecz zespołu programistycznego. Jest to całkiem przewidywalne, programiści powinni być w stanie samoorganizować się i przeprowadzać spotkania codzienne, bez pomocy Scrum Mastera i Product Ownera.

Rys. 5.16 Uczestnicy Planowania sprintu

Rys. 5.17 Uczestnicy Przeglądu sprintu

Rys. 5.18 Uczestnicy Retrospektywy sprintu

Rys. 5.19 Uczestnicy Daily

Kolejna seria pytań od pytania 2.7 do pytanie 2.10, dotyczyła długości trwania poszczególnych spotkań. W pierwszej kolejności patrząc na Rys. 5.20, który dotyczy czasu trwania spotkania planowania sprintu. Zauważyć można, że najwięcej odpowiedzi waha się między 1-2 godzinami. Zastanawiające są odpowiedzi skrajne poniżej pół godziny i powyżej czterech godzin. Na kolejnym rysunku 5.21 widzimy rozkład dla spotkania przeglądu sprintu. Przegląd sprintu, również waha się między 1-2 godzinami, jednak z naciskiem na 1 godzinę. Jest to już trochę zbyt krótko na przeprowadzenie odpowiedniego przeglądu, jednak bardzo dużo zależy od projektu i zespołu. Następnie patrząc na Rys. 5.22 dotyczący czasu trwania retrospektywy sprintu, widzimy, że jak i poprzednio czas oscyluje pomiędzy 1-2 godzinami, jednak również jak można zauważyć 1/3 ankietowanych pominęła to pytanie. Na końcu przeanalizujmy Rys. 5.23 dotyczący daily. Nie jest to oczywiście zaskoczenie, że trwa ono najczęściej 15 minut lub mniej, jest to jak najbardziej zgodne z metodyką Scrum i zalecane.

Rys. 5.20 Czas trwania spotkania planowania sprintu

Rys. 5.21 Czas trwania spotkania przeglądu sprintu

Rys. 5.22 Czas trwania spotkania retrospektywy sprintu

Rys. 5.23 Czas trwania daily

Pytanie 2.11 poruszało kwestię osoby odpowiedzialnej za ustalanie Backlogu. Wyniki przedstawione zostały na Rys. 5.24, jak widać zdecydowana większość odpowiedzi, bo aż 70% to Product Owner, jednak pojawiały się też inne role. Odpowiedzi na potrzeby wykresu zostały uproszczone jednak Analityk współpracował czasem Project Managerem, Klient z developerami, Project Manager

z developerami i klientem, w jednym przypadku Project Manager rozdzielał już od razu zadania na sprinty. Natomiast Product Owner współpracował z analitykiem, zespołem programistycznym, testerami, z Project Managerem, z Scrum Masterem, ze wsparciem, z Team Leadem oraz w jednym przypadku był to Product Owner od klienta.

Rys. 5.24 Osoba ustalająca Backlog

Pytanie 2.12, było dwuczłonowe, na Rys 5.25 widzimy odpowiedz na pierwszą jego część „Czy Backlog jest często zmieniany?”, natomiast na Rys 5.26 widzimy rozkład danych na odpowiedz „Przez kogo był zmieniany?”. Na Pierwszym wspomnianym wykresie dokładnie widać, że Backlog zmienia się dynamicznie w 73% zespołów i bardzo dobrze to świadczy o zespołach, jednak pojawiło się 17% odpowiedzi „nie”, co bardziej zastanawiające, w niektórych z nich podkreślone było, że backlog jest ustalony przez klienta i nie może być zmieniany. Przechodząc do drugiej części pytania i kolejnego wykresu, widzimy, że Backlog zmieniany był głównie przez Product Ownera, co również jest dobrą informacją, ponieważ to on jest odpowiedzialny za jego pielęgnację. W odpowiedziach, zawarte były również informacje, że Product Owner często korzystał z pomocy zespołu lub Project Managera.

Rys. 5.25 Zmiany w Backlogu

Rys. 5.26 Osoba odpowiedzialna za zmiany w Backlogu

Pytanie 2.13 poruszało temat wykorzystywania metody opisywania elementów Backlogu za pomocą User Stories i uszczegóławiających kryteriów akceptacji. Jednak, na Rys. 5.27 widać, że nie jest to aż tak popularna metoda. Metoda ta jest oczywiście opcjonalny i nie opisana w Scrum Guide, jednak bardzo często preferowana. W tym przypadku niewiele więcej niż 1/3 ankietowanych opisuje zarówno historyjki użytkownika, jak i ich kryteria akceptacji. Natomiast prawie połowa ankietowanych nie korzysta z tej metody w ogóle.

Rys. 5.27 User Stories

Pytanie 2.14 dotyczyło określania Definition of Done dla Backlogu. Jak widzimy na Rys. 5.28 prawie połowa zespołów nie opisywała Definition of Done. Oznacza to, że programiści tak naprawdę nie mieli dokładnie określonego, kiedy ich zadanie jest ukończone. Oczywiście nie jest to konieczne, ale bardzo ułatwia pracę, usuwa niedopowiedzenia i zmniejsza prawdopodobieństwo wystąpienia błędów.

Rys. 5.28 Definition of Done

Pytanie 2.15 dotyczyło użycia metody MoSCoW, jest to metoda kategoryzacji elementów Backlogu na zadania, które muszą wejść w zakres projektu (Must), te które powinny (Could), te które dobrze by było (Should) i tzw. „wodotryski” (Want), czyli rzeczy, które klient chce, ale ich niekoniecznie potrzebuje. Jednak jak widzimy na Rys. 5.29 nie jest to popularna metoda i wykorzystywana była tylko

w 19% projektów. Pod odpowiedziami „inne” znajdowały się odpowiedzi czasami i było na początku, ale później zrezygnowaliśmy.

Rys. 5.29 MOSCOW

Pytanie 2.16 poruszało zagadnienie samoorganizacji zespołu. Na Rys. 5.30 widzimy zobrazowane wyniki. Jak można zauważyć, w zdecydowanej większość, bo w 76%, zespół był samoorganizujący się. Jednak, aż 22% zespołów miały zadania narzucane przez Dyrektor Departamentu, Team Leadear, Project Managera, Product Ownera lub Scrum Mastera.

Rys. 5.30 Samoorganizacja

Pytanie 2.17 dotyczyło nadawania priorytetów do zadań oraz osoby za to zadanie odpowiedzialnej. Wyniki do pierwszego pytania zostały przedstawione na Rys. 5.31 i jak można zauważyć zdecydowana większość, bo aż 85% zespołów nadaje priorytety do zadań, jest to dobra praktyka, która pomaga przy planowaniu. Odpowiedzi na drugie pytanie przedstawione są na Rys. 5.32 i tutaj również zaskoczeniem nie będzie, że najczęściej za priorytety jest odpowiedzialny Product Owner. Project Manager korzystał często z pomocy Team Leadera, Tech Leade’a, zarządu lub zespołu. Natomiast Product Owner pracował z Architektem, Project Managerem, Scrum Masterem, Team Leadem, zespołem lub CTO. Pod kolumną „inne” kryją się odpowiedzi tj. analityk, architekt, dyrektor departamentu, klient, Leader, osoba tworząca ticket.

Rys. 5.31 Piorytetyzacja zadań

Rys. 5.32 Nadawanie priorytetów

Pytanie 2.18 dotyczyło ustalania czasochłonności zadań oraz osoby za to zadanie odpowiedzialnej. Na Rys. 5.33 widać zobrazowane odpowiedzi na pierwsze pytanie i aż 80% zespołów stara się szacować swoje zadania w Backlogu. Przenosząc się do Rys. 5.34 można ocenić, kto najczęściej szacuje dane zadania. Zdecydowana większość odpowiedzi odnosi się do zespołu lub samych programistów. Najczęściej szacowania odbywały się w story pointach lub scrum pokerze. W odpowiedziach „inne” znajduje się Doradca techniczny, Kierownik Projektu, Leader Techniczny, Team Leader, Product Owner, Scrum Master i CTO.

Rys. 5.33 Szacowanie zadań

Rys. 5.34 Ustalanie czasochłonności

Pytanie 2.19 dotyczyło wielkości zespołu programistycznego, liczby zostały przedstawione na Rys. 5.35, natomiast określenie norm przedstawione jest na Rys. 5.36. Jak widać najpewniej ponad 80% zespołów pracuje w nie za dużej i nie za małej grupie, jednak wciąż zastanawiające są grupy skraje po

2 lub 35 osób. Metodyka Scrum nie powinna być prowadzona ani dla tak małych, ani dla tak dużych zespołów.

Rys. 5.35 Wielkość zespołów programistycznych

Rys. 5.36 Zgodność z wytycznymi wielkości zespołów

Pytanie 2.20 poruszało temat pracy programistów w jednym projekcie lub rozdrabnianiu się na wiele. Przedstawiony na Rys. 5.37 wykres pokazuje, że ponad 1/3 zespołów składa się z programistów zajmujących się innymi zagadnieniami. Rozbija to bardzo pracę i nie jest zalecaną praktyką. Ponieważ trudniej jest zaplanować pracę, gdy programiści mogą być w niektórych sprintach pochłonięci innym projektem.

Rys. 5.37 Praca przy projektach programistów

Pytanie 2.21 poruszało podobno kwestie, mianowicie etat pracowników w projekcie. Odpowiedzi przedstawione na Rys. 5.38, są jednak bardziej zadowalające. 96 % zespołów składa się z pracowników pracujących na pełen etat lub przynajmniej na większość etatu.

Rys. 5.38 Etat pracy programistów

Pytanie 2.22 odnosiło się do zasady metodyki Scrum, mówiącej, że nie powinno się określać ról dla członków zespołu programistycznego. Jednak jak można zauważyć na Rys. 5.39 rolę są często określane. Może to być kwestia przyzwyczajenia, jednak powinno się starać, by każdy programista był członkiem zespołu z danym zestawem umiejętności. Ponieważ nie tworzy to sytuacji narzucania niektórym członkom zespołu zadań z powodu ich roli i zachęca do podejmowania się różnych zagadnień. Tworzymy w ten sposób mulitidyscyplinarny zespół.

Rys. 5.39 Określanie ról w zespole

Pytanie 2.23 dotyczyło niezależności, względem innych zespołów, zespołu programistycznego. Jak możemy zauważyć na Rys. 5.40 prawie połowa zespołów, potrzebuje innych zespołów do pracy. Jest to duży problem i często wąskie gardło, gdzie zatrzymuje się projekt.

Rys. 5.40 Samodzielna praca zespołu

Pytanie 2.24 dotyczyło tworzenia wykresu wypalania dla projektu. Wyniki przedstawione na Rys. 5.41 pokazują, że tylko niecała 1/3 zespołów tworzy i analizuje wykres wypalania. Nie jest to co prawda praktyka opisana w Scrum Guide, ale jest bardzo preferowana. Wykres wypalania pomaga

w utrzymywaniu transparentności i analizie szybkości zespołu.

Rys. 5.41 Wykres wypalania

Pytanie 2.25 było pytaniem kontrolnym, w celu sprawdzenia czy aby na pewno Scrum Master

i Product Owner w zespole to nie jest jedna osoba. Jak widzimy na Rys. 5.42, tylko 10% zespołów łączy te role, a u 89% istnieją one jako osobne byty. Co bardziej zastanawia w jednej odpowiedzi, ktoś nie był

w stanie tego określić.

Rys. 5.42 Wspólna rola

Pytanie 2.26 dotyczyło istnienia dodatkowych ról, nie tych proponowanych przez Scrum Guide. Jak widzimy na Rys. 5.43 ponad połowa zespołów posiadała dodatkowe role. Wykres

z wymienionymi rolami widać na Rys. 5.44 i najczęściej był to Project Managera, czyli rola, która najpewniej istniała przed przejściem na metodykę Scrum i nie została ani przekształcona, ani usunięta. Dodatkowe role są często wymagane przez biznes lub kulturę organizacji, jednak powinno się tego zdecydowanie unikać. W kolumnie „inne” na Rys. 5.44 zawierają się odpowiedzi tj. Chapter Lead, Guilds Lead, Tribe Lead, Delivery Manager, Engineering Manager, design lead, System Manager, Sponsor, Stakeholder, Project Quality Manager, Product Manager, Project Lead, Production Manager (Planner), Sales Leader, Service Design, Test Manager, Release Manager, Liderzy Technologii i Agile Coach.

Rys. 5.43 Dodatkowe role

Rys. 5.44 Rodzaje dodatkowych ról

Pytanie 2.27 analogicznie odnosiło się do dodatkowych spotkań, nie określonych przez Scrum Guide. Na Rys. 5.45 widzimy, że ponad połowa zespołów posiadała dodatkowe spotkania. Natomiast Rys. 5.46 przedstawia nam rodzaje tych spotkań, zazwyczaj był to Backlog Refainment, spotkania synchronizujące i demo z klientem. Wszystkie te spotkania są istotne dla dobrego funcjonowania metodyki Scrum w projekcie. W kolumnie „inne” znalazły się takie odpowiedzi jak Code review, dev leaders meeting, dev meeting, Discover Weekly (wspólne weryfikowanie aplikacji, szukanie potencjalnych problemów), doszkalające u klienta, Improvement Meeting, Miesięczne progress update'y, cotygodniowe feature update'y, organizacyjne, motywujące, spotkania działów, spotkania monitorujące, spotkania przyrostowe z klientem, spotkanie przygotowujące przed przeglądem, warsztaty analityczne, wprowadzające

i spotkanie podzespołów z głównym Analitykiem.

Rys. 5.45 Dodatkowe spotkania

Rys. 5.46 Rodzaje dodatkowych spotkań

Ostatnie pytanie z sekcji drugiej 2.28 dotyczyło narzędzi wspomagających prace w metodyce Scrum. Wyniki zostały przedstawione na Rys. 5.47 i jak można zauważyć najpopularniejszym z nich jest zdecydowanie Jira, a komunikacyjnie Slack. W kolumnie „inne” zostały zawarte odpowiedzi hipchat, hangout, whatsapp, rqm, azureDevOps, facebook, svn, Oracle ebs, ms project, funretro, rally, kanbanize, doodle, ankiety gogle, ms planner, xray, jabber, roadmunk, telegram, aha, retrobot, youtrack, glip, openproject, lync, red hat, zammad, clickUp, tfs board, project Server.

Rys. 5.47 Narzędzia

Trzecia i ostatnia sekcja, dotyczyła subiektywnych ankietowanego w pracy w metodyce Scrum. Pierwsze pytanie 3.1 poruszało stosunek do daily. Wyniki zostały przedstawione na Rys. 5.48, jak widzimy aż 73 % ankietowanych widziała same zalety, podkreślano możliwość rozwiązania problemów, polepszenie komunikacji i świadomości członków zespołu o projekcie, możliwość wypowiedzenia się oraz możliwość przygotowania się na najbliższe godziny pracy. W skomplikowanych odpowiedziach, zawierały się stwierdzenia niejednorodne. Ankietowani chwalili spotkanie, ale w wypowiedziach pojawiały się stwierdzenia, że synchronizacja jest przydatna, jednak nie powinno to być rozwiązywane na daily, że zespól wyciągał coś ze spotkań, ale osoba ankietowana osobiście nie oraz co najważniejsze, że spotkanie jest przeciągane. Natomiast osoby negatywnie nastawione do daily, poruszały zagadnienia tj. niewykorzystywanie potencjału, pomijanie dyskusji, daily dla wielu projektów i obecność osoby, która nie uczestniczy w nich jest zbędna, tracenie czasu, bezsensowność, bo zespół siedzi razem w pokoju

i rozmawiają, że wystarczyłoby jedno na 2 dni, że jest spotkaniem mocno statusowym, czasem nie dużo wnosi (rozmawianie o rzeczach nie związanych z projektem, raportowanie braku postępów). Co ciekawe, aż 7% zespołów z 13% wypowiadających się negatywnie o daily, nie miało Scrum Mastera, co mogło powodować złe przeprowadzanie spotkania, gubienie się w dyskusjach i przedłużanie.

Rys. 5.48 Stosunek do daily

Drugie pytanie 3.2 dotyczyło stosunku do roli Scrum Mastera. Wyniki zostały przedstawione na Rys. 5.49 i jak widać ponad 1/3 osób pominęła to pytanie, czasami było to związane z nie istnieniem roli Scrum Mastera lub tym, że Scrum Master uzupełniał ankietę i nie chciał oceniać sam swojej pracy. Skupiając się jednak na początek na pozytywnych odpowiedziach, ankietowani podkreślali pomoc

w usuwaniu przeszkód lub uczenie zespołu jak je usuwać, pilnowanie zespołu, pomaganie w zrozumieniu ceremonii scrumowych, reagowanie na zmiany, bycie pomocnym, dostosowywanie metodyki Scrum do potrzeb zespołu, skupianie się na efektach i zespole, nie na przestrzeganiu ślepo reguł metodyki Scrum, poprawianiu komunikacji, spajaniu zespołu, dbaniu o dobre morale, organizowaniu i przeprowadzaniu spotkania, pracy nie tylko z zespołem, ale również wpieraniu organizacji we wdrażaniu metodyki Scrum. Przechodząc do odpowiedzi o stopniu skomplikowanym, ankietowani zwracali uwagę na uległość Scrum Mastera, w stosunku do klienta i zarządu, skrajne przestrzeganie metodyki Scrum, zbyt dokładne pilnowanie zasad, posiadanie przez Scrum Mastera innej dodatkowej roli i nie poczuwaniu się do bycia Scrum Masterem, kiepskie prowadzenie spotkań, bycie ciężko uchwytnym, jak i również często używano sformowania „stara się”. Negatywnych wypowiedzi pojawiło się 9% i podkreślały one wywieranie presji

i narzucanie rozwiązań zespołowi, niewywiązywanie się ze swoich obowiązków, czasem był to Kierownik, wprowadzanie zamieszania oraz jedna opinia, że rola ta jest zbędna i zespół jest w stanie sam pilnować ceremonii (co tylko potwierdza dojrzałość i samoorganizacje zespołu).

Rys. 5.49 Stosunek do Scrum Mastera

Ostatnie pytanie 3.3 poruszało temat oceny do metodyki Scrum. Wyniki zostały przedstawione na Rys. 5.50 i jak można zauważyć prawie połowa ankietowanych wypowiada się o metodyce Scrum pozytywnie. Podkreślają oni dobrą organizację pracy, mniejszy stres, gdy zadania na dany sprint, są określone a nie podrzucane z dnia na dzień, zauważają pozytywne aspekty planowania, każdy znał swoją role i obowiązki, podkreślali dobre dopasowania metodyki Scrum do projektu i zespołu, pomoc

w komunikacji, możliwość zmian, przejrzystość w projekcie, pomoc w dostarczaniu lepszego produktu oraz ciągłe dążenie do doskonałości. Przechodząc do określeń „skomplikowane” ankietowani zwracali uwagę na zbyt restrykcyjne trzymanie się zasad, bezsensowną estymację, przerost ceremonii scrumowych, wtrącenia Scrum Mastera burzące pracę, Product Owner i Analityk, którzy nie uczestniczyli w spotkaniach, nie przekonani do metodyki Scrum programiści, brak możliwości określenia deadline’u biznesowi, bycie nakładką na Waterfalla, zbyt dużo „wrzutek” zadań, biznes nie rozumiejący metodyki Scrum, zbyt wiele planowane na jeden sprint, za wolne, zbyt duży zespół, nie rozumienie wartości scrumowych oraz oczywiście nie przestrzeganie metodyki Scrum w 100%. Kończąc na negatywnych opiniach, był to projekt, gdzie metodyka Scrum nie sprawdziła się, jednak była to wina klienta, który nie potrafił wdrożyć metodyki Scrum po swojej stronie. Inni zarzucali zbyt dużo formalności i obecność wszystkich członków zespołu na ciągłych spotkaniach oraz to, że metodyka Scrum nie działa, gdy programiści są w wielu projektach. Ostatnim dużym zarzutem było to, że role najbardziej odpowiedzialne w zespole sobie z nim nie radzą i go nie rozumieją.

Rys. 5.50 Stosunek do metodyki Scrum

Przeprowadzone przeze mnie wywiady dotyczyły grupy 20% ankietowanych, byli to zarówno studenci jak i absolwenci. W grupie przeważali programiści, jednak pojawił się również Product Owner i Scrum Master. Bezpośrednie rozmowy pozwoliły mi na dokładniejsze zrozumienie przedstawianej przez ankietowanych adaptacji metodyki Scrum, byli oni w stanie wytłumaczyć jak i dlaczego dochodziło do łamania i modyfikacji zasad określonych w Scrum Guide. Ankieta w formie bezpośredniego wywiadu pozwoliła mi również na dokładne wyjaśnienie pytań osobie ankietowanej, dlatego mogłam mieć pewność, że odpowiedzi na pytania są dokładne i poprawne.

Przenalizowałam, ile ankietowanych korzystało z metodyki Scrum zgodnie z zasadami określonymi w Scrum Guide i okazało się, że było to zaledwie 9% zespołów. Jednak nawet w tych zespołach pojawiały się sytuacje, gdzie role były określone z góry i zespół potrzebował innych zespołów do wytwarzania oprogramowania. Zwróciłam uwagę również na opinie tych 9 % dotyczące oceny metodyki Scrum i były mieszane, nie wypowiadali się negatywnie jednak mieli zastrzeżenia do nadmiaru ceremonii i przerostu formalności.

Analizując przypadki, gdzie metodyka Scrum nie była wprowadzona zgodnie z zasadami, ale zachowywane były najważniejsze zasady wyliczyłam 44% zespołów. W najczęstszych odstępstwach pojawiało się:

1. Pojawiające się „wrzutki” zadań podczas trwania sprintu (84%)

2. Nietworzenie wykresu wypalania lub nieanalizowanie go (75%)

3. Sprint nie kończący się ukończoną funkcjonalnością (56%)

4. Programiści pracujący w wielu projektach (45%)

5. Brak określonego Definition of Done (38%)

6. Potrzeba współpracy z innymi zespołami w celu dostarczenia rozwiązania (38%)

7. Określanie ról w zespole (27%)

8. Nieokreślanie celu sprintu (27%)

9. Nie zmieniający się Backlog (11%)

10. Brak szacowania czasochłonności (9%)

W tej grupie 85% ankietowanych wypowiadało się pozytywnie na temat daily, 7% negatywnie (min. byli w jednym pokoju lub osoba, które nie jest zaangażowana w ten projekt musiała uczestniczyć), 4% skomplikowanie (zwrócenie uwagi na przedłużanie) i 4% nie odpowiedziało. Na temat Scrum Mastera 62% wypowiedziało się pozytywnie, 4% negatywnie (min. jest bardziej koordynatorem projektu lub programista, który konsultuje się z Product Ownerem), 18% skomplikowanie (min. Zbyt restrykcyjne pilnowanie zasad, małe doświadczenie, przedłużanie spotkań) i 16% nie odpowiedziało. Jeśli chodzi

o samą ocenę metodyki Scrum 48% pozytywnie, 16% negatywnie (min. ciągłe spotkania, nie wykorzystywanie potencjału, zarząd nie rozumiejący Agile i zbyt duże skupienie na metodyce Scrum, zamiast na wytwarzaniu oprogramowania), 25% skomplikowanie (min. nie przekonani programiści, za dużo spotkań i ceremonii, brak zrozumienia ze strony biznesu, za dużo wrzutek i nie do końca trzymanie się wszystkich zasad metodyki Scrum) i 11% nie udzieliło odpowiedzi.

W ostatniej grupie 47% nie pracujących zgodnie ze metodyką Scrum, pojawiły się rażące odstępstwa takie jak:

1. Brak Scrum Mastera (51%)

2. Brak Product Ownera (34%)

3. Brak i Scrum Mastera i Product Ownera (19%)

4. Scrum Master i Product Owner to jedna osoba (23%)

5. Brak spotkania przeglądu sprintu (12%)

6. Brak spotkania retrospektywy sprintu (29%)

7. Brak daily (10%)

8. Zmienny czas trwania sprintu (12%)

9. Brak Backlogu sprintu (15%)

10. Zbyt duży zespół programistyczny (21%)

Jednak mimo wszystko 45% opowiedziało się pozytywnie za metodyką Scrum, 11% negatywnie (min. klient nie współpracował i odpowiedzialne osoby nie rozumiały idei Agile), 36% mało mieszane uczucia (min. trzymanie się zasad scrumowych, męcząca estymacja, zbyt duży nacisk na metodykę Scrum, Scrum Master przeszkadzający w pracy, nieodpowiednie wdrożenie metodyki Scrum i brak zrozumienia idei zwinnego wytwarzania oprogramowania) i 8% nie udzieliło odpowiedzi.

Wychodzą z tego dość proste wnioski – nie powinniśmy „robić scruma dla scruma”, tylko skupiać się na wynikach, zadowoleniu pracowników i efektach naszej pracy. W wielu sytuacjach również problem, z metodyką Scrum stoi po stronie Scrum Mastera – nieistnienia w zespole takiej roli lub przypisania tej roli do osoby nie kompetentnej. Więcej na temat pracy Scrum Mastera i raportu „Scrum Master Trends 2019”, można znaleźć w artykule [26].

Oczywiście należy też pamiętać, że wszystkie zalecenia metodyki Scrum, nie pojawiły się w Scrum Guide bez przyczyny, dlatego też złą praktyką jest odrzucanie elementów zaproponowanych przez metodykę bez próby ich implementacji. Nie zawsze wszystko udaje się za pierwszym razem, a metodyka Scrum zakłada dążenie do ciągłej doskonałości i próbowaniu nowych metod i technik. Przytoczę znaną metodę uczenia się sztuk walki Shu Ha Ri, która znajduje odwzorowanie również w nauce Agile. Jest to podejście dzielące naukę nowej metodyki na 3 etapy:

1) Słuchanie – to powtarzanie, uczenie się nowych technik, słuchanie nowych rozwiązań, często nie rozumiejąc jeszcze zasad i reguł za nimi stojących. Dlatego też, oczekiwane jest, by w tym etapie uczniowie przestrzegali technik bez odstępstw.

2) Odłączenie – gdy uczeń zrozumie już zasady, zaczyna eksperymentować z różnymi technikami, łączy je lub zaczyna korzystać w innych źródeł.

3) Opuszczenie – w tym stadium uczeń głęboko rozumie zasady i ich konsekwencje. Uczeń staje się nauczycielem tworzącym własne techniki i rozwiązania.

Odnosi się do to wdrażania meotdyki Scrum, na początku powinniśmy przestrzegać wszystkich zasad zgodnie z opisem w Scrum Guide, jednak z czasem powinniśmy zacząć eksperymentować, dostosowywać się, aż będziemy na tyle dojrzali, że będziemy w stanie sami proponować nowe rozwiązania.

6. Wnioski

Opierając się na własnych przemyśleniach, doświadczeniu zebranym podczas studiów, w pracy

w metodyce Scrum, rozmowach z ankietowanymi i przeglądów literatury, wypracowałam listę dobrych praktyk w metodyce Scrum:

1. Wdrażanie metodyki Scrum. Po pierwsze proces wdrażania metodyki Scrum, powinien być dobrze przeanalizowany przez wdrażającą organizację. Nie może to być zmiana tylko jednego zespołu, tylko strategii całej firmy. Należy mieć świadomość, że jest to proces długi i kosztowny. Firma musi być na niego dobrze przygotowana i zgadzać się na reorganizacje procesów nie zgodnych z metodyką Scrum. [5, 8, 10, 21]

2. Dobór metodyki do projektu. Metodykę Scrum można wdrożyć oczywiście do każdego projektu, jednakże istnieją projekty, w których sprawdza się gorzej i powinno się wziąć pod uwagę, czy nie lepsza byłaby inna metoda zwinna. Przykładowo w projektach utrzymaniowych, gdzie musimy reagować na zmiany na bieżąco, gdy specyfika firmy jest taka, że programiści pracują jednocześnie przy wielu projektach lub gdy zespół jest za duży. Nad metodyką Scrum powinniśmy się zastanowić, kiedy pojawiają się fizyczne bariery tj. rozproszony zespół po innych strefach czasowych. Problemem stanowi również niedoświadczony zespół programistyczny. [4, 21, 33]

3. Wybór elementów metodyki Scrum. Bardzo częstym błędem jest usuwanie elementów, które nam nie pasują, te najczęściej usuwane opisane są w osobnych punktach, jednak chciałabym zwrócić uwagę na fakt, że powinno jak najdłużej dążyć się do wprowadzania nakazów metodyki Scrum. Nie wszystko działa od samego początku i jeśli nie działa oznacza to zazwyczaj, że jest źle zaimplementowane i powinniśmy spróbować innego sposobu. W metodyce Scrum chodzi

o dopasowywanie się, nie o wybieranie pasujących nam elementów. [32]

4. Metodyka Scrum jako rozwiązanie problemów organizacji. Należy pamiętać, że metodyka Scrum jest nowym sposobem myślenia, nie rozwiązaniem problemów. Wdrożenie metodyki Scrum nie spowoduje nagle ze znikną wszystkie problemy, jednak firmy często dokładnie tego oczekują. Jeśli problem istniał, to będzie najpewniej istniał dalej, co najwyżej metodyka Scrum ukaże nam miejsce, w którym występuje. [32]

5. Estymacje. Często problematyczne okazują się szacowania, programiści uznają je za zbędne i są niezadowoleni, gdy są do nich zmuszani. Jednak często problemem jest właśnie to, że traktowane są one jako coś pewnego i ustalonego. Estymowanie czasu jest bardzo ważne dla klienta, jednak

i klient, i programiści powinni wiedzieć, że są one prognozami i szacowaniami. Różne sposoby na poprowadzenie esytmat opisane są w książce [17]. [28]

6. Problem „niedziałającego Scruma”. Napotkać można stwierdzenie, że „Scrum nie działa”, jednak powinniśmy zastanowić się, czy to metodyka Scrum nie działa, czy też może my nie robimy czegoś źle, a metodyka nam właśnie bardzo dobrze pokazuje, w którym miejscu popełniamy ten błąd.

7. Aplikacje o dużym stopniu trudności. Nie powinno się narzucać zespołowi skomplikowanych narzędzi i oprogramowania. Przypominając sobie manifest Agile „Ludzie i interakcje od procesów i narzędzi”. Narzędzia mają nam pomagać, a nie przeszkadzać.

8. Brak inspekcji i adaptacji. Błędem jest rezygnowanie z inspekcji, adaptacji lub obu. Metodyka Scrum opiera się na empiryzmie i rezygnując z popraw i miejsc do zastanowienia się nad prowadzeniem metodyki, narzucamy w ten sposób kolejny metodę typu kaskadowego, która nie adaptuje się do zmian, których ewidentnie potrzebujemy. Zespół nie będzie zadowolony

z metodyki Scrum, jeśli będziemy poprawiać rzeczy, które nie działają. [10]

9. Rola kierownika. Musimy pamiętać, że w metodyce Scrum nie istnieje rola kierownika, zespół jest samoorganizujący się i posiada władzę. Nie można pozwolić, by kierownik lub Scrum Master narzucał rozwiązania lub rozdzielał zadania. Może on proponować i przekonywać do pewnych rozwiązań, jednak to od zespołu należy ostateczna decyzja. [33]

10. Wymagania projektowe. W metodyce Scrum nie ma miejsca na zdefiniowane z góry wymagania projektowe, klient nie może wymagać pracy na zdefiniowanych, szczegółowych, niezmiennych wymaganiach. [32]

11. Wartości metodyki Scrum. Metodyka Scrum opiera się na zespole wyznającym grupę wartości, opisanych w rozdziale 4. Wszyscy powinni być ich świadomi i powinny one być czynnikiem działań. [10]

12. Planowanie. Czasami za duży nacisk jest kładziony na planowanie. Jest to oczywiście niezbędny element metodyki Scrum i nie możemy go pomijać, ale musimy pamiętać, że planowanie ma nam pomagać, a nie przeszkadzać. Nie jesteśmy w stanie zaplanować każdej najmniejszej rzeczy oraz nasze szacowania i plany mogą się zmieniać. [8, 34]

13. Ustalona hierarchia w firmie. W punkcie 1 poruszony został temat gotowości organizacji na zmianę, jednak należy również wziąć pod uwagę hierarchię w firmie i sposób postępowania sprzed zmiany. Jeśli ludzie mają głęboko zakorzenioną drabinę odpowiedzialności w pracy, mogą bać się sami podejmować decyzję i wciąż zgłaszać po pomoc do zwierzchników. Przed takim samym problemem stają osoby nadzorujące, by przestawić się na to, że nie mają już takiego wpływu na pracę nad projektem. [8]

14. Kopiowanie rozwiązań. W metodyce Scrum ważna jest nauka na doświadczeniu i błędach, również innych ludzi. Świetnie jest czerpać z doświadczenia innych jednak nie powinniśmy ślepo kopiować po innych osobach. To, że u kogoś działa jeden sposób nie oznacza, że będzie działał

u nas. Powinniśmy sami dojść do optymalnego rozwiązania, jedynie wzorując się na innych przykładach. [10]

15. Nowe reguły pracy. Metodyka Scrum zachęca do próbowania nowych rzeczy i otwartego umysłu. Często problem pojawia się, gdy ludzie się zamykają na jeden pomysł - „My mamy swoją metodę pracy, sprawdzała się we wszystkich projektach, nie ma potrzeby jej zmieniać”. [13]

16. Podział zadań. Problem pojawia się, kiedy automatycznie zaczynamy przydzielać zadania wg. klucza. Ta osoba zajmuje się tym zagadnieniem, a ta tym. Metodyka Scrum zachęca do posiadania zestawu umiejętności i zespołów cross-funkcjonalnych. [13]

17. Niechęć. Co najważniejsze, problem, którego nie jesteśmy w stanie przeskoczyć to moment,

w którym nikt w projekcie metodyki Scrum nie chce. Jeśli wszyscy są negatywnie nastawieni do używania metody, najpewniej obojętne co zrobimy, nie będziemy w stanie przekonać ludzi do tego, że jest ona dobra. Bardzo dobra praktyką jest przeprowadzanie szkoleń, zanim metodyka Scrum zostanie wprowadzona, pozwala to na lepsze zrozumienie wartości i zasad przez członków zespołu. Wszyscy członkowie zespołu, powinni rozumieć czym metodyka Scrum jest i rozumieć po co są poszczególne spotkania. Jeśli ktoś nie wie po co jest na danym spotkaniu budzi to w nim irytację i niechęć. [8,10,21, 31,34]

18. Błędne koło. Często możemy wpaść w błędne koło metodyki Scrum, gdzie Product Owner obiecuje klientowi wiele rzeczy, których zespół programistyczny nie jest w stanie dostarczyć. Zostaje zaplanowane zbyt dużo zadań na jeden sprint, developerzy starając się wyrobić w czasie tworzą produkt o gorszej jakości. Product Owner jest niezadowolony z powodu obniżenia jakości i na Scrum Masterze wywierana jest presja, która przechodzi na developerów. Sytuacja wymusza zatrudnienie dodatkowych developerów i architektów, gdzie oczywiście zwiększa to koszty i i tak produkt nie jest tworzony szybciej. Na koniec nikt nie jest zadowolony i wszyscy uważa, że jest to wina metodyki Scrum. [14]

19. Sposób myślenia. Należy pamiętać, że nie pracujemy w agile, tylko jesteśmy agile. Agile to nie metodyka, tylko sposób myślenia, na który musimy przestawić zespół i biznes. [5]

20. Zdrowy rozsądek. Metodyka Scrum to receptura, do której musimy się dostosować. Powinniśmy robić rzeczy dlatego że mają one sens, a nie dlatego że tak mówi Scrum Guide. Zawsze na pierwszym miejscu powinien być głos rozsądku. [4]

21. Wielkość zespołu. Przyjęcie zbyt dużego lub zbyt małego zespołu. Zbyt duży zespół będzie utrudniał sprawną i szybką komunikację, przedłużał spotkania i skutkował brakiem pełnego zaangażowania wszystkich członków (nie wszyscy będą słuchali i odzywali się na spotkaniach). Natomiast zbyt mały zespół spowoduje problemy z dowożeniem produktu, najczęściej będzie to zbyt mała liczba programistów, co zwiększa ryzyko opóźnień w momencie ich urlopów. [21, 31]

22. Zmienny sprint. Posiadając jedną długość sprintu jesteśmy w stanie określać zdolność zespołu na realizację zadań. Jeśli długość sprintu nie jest stała, utrudnia nam to planowanie. [31]

23. Nieobecności na spotkaniach. Metodyka Scrum opiera się na komunikacji i zaufaniu. Gdy ludzie zaczynają omijać spotkania ani nie wiedzą co dzieje się w projekcie, ani nie komunikują czym się aktualnie zajmują. W metodyce Scrum każdy ma takie samo prawo głosu, dlatego powinien je wykorzystywać. [31]

24. Wrzutki zadań podczas sprintu. Określona ilość zadań jest dla programistów zapewnieniem, że pracują w środowisku bez stresowym. „Wpadające” zadania rozbiją pracę i czasem powodują nadgodziny. Ustalone zadania są pewnym rodzajem niepisanej umowy, między programistami,

a Product Ownerem. Programiści obiecują ukończyć wszystkie zadania, ale Product Owner obiecuję, że to jedyne co mają dostarczyć. [33]

25. Role. Rezygnowanie ze Scrum Mastera, Product Ownera lub łącznie tych ról. Obie role istnieją

w metodyce Scrum nie bez przyczyny i mają swoje określone zadania i obowiązki. Role te tworzą idealnie zbalansowany zespół, gdzie Product Owner dba o dobro klienta, a Scrum Master o dobro zespołu. Rezygnując z którejś z nich zaburzany ten balans. Tak samo, jeśli połączymy te role

i zrzucimy na jedną osobę wszystkie te obowiązki. Możemy spodziewać się, że zaniedba jedną

z nich. [4, 33]

26. Retrospekcja. Bardzo często zespoły rezygnują z przeprowadzania retrospektywy lub robią ją niedbale. Spotkanie to istnieje po to, żebyśmy zatrzymali się i pomyśleli co nam nie wychodzi lub co mogłoby nam iść lepiej. Jeśli nie staramy się poprawiać i dążyć do doskonałości, powielamy błędy, które popełniamy i gorzej nam się pracuje. [28, 33]

27. Cel projektu. Skupienie się na celach krótkoterminowych i gubienie całościowego obrazu. Czasem skupiając się tylko na celach poszczególnych sprintów, potrafimy stracić cel całego projektu. Dlatego też, ważna jest transparentność w projekcie i świadomość wszystkich członków zespołu

o wyznaczonych celach. [32]

28. Daily. W niektórych sytuacjach zespołu rezygnują z daily. Jest to spotkanie, gdzie możemy dowiedzieć się czym kto aktualnie się zajmuje i zaplanować pracę dla siebie na najbliższe 24h. Jeśli rezygnujemy z takich spotkań, mogą pojawiać się problemy z komunikacja i transparentnością. [28]

29. Definition of Done. Niezwykle ważne jest, by programiści wiedzieli, kiedy zadanie, które wykonują jest skończone. Jeśli nie mamy zapewnionej tej wiedzy będzie to powodowało określanie zadań przez programistów jako ukończonych, gdy nie są one jeszcze ukończone. [28]

30. Losowo przydzielony Scrum Master. Wiele zespołów nie docenia roli Scrum Mastera. Uważa, że wystarczy przeszkolić szybko programistę i postawić go, by „pilnował przestrzegania metodyki Scrum”. Rola Scrum Mastera, jest bardzo złożona i nie może jej przyjąć byle kto. Zawsze powinniśmy się dobrze zastanowić kogo postawić na tej roli i często może nawet zatrudnić nową osobę na takie stanowisko. [28, 34]

Więcej na temat na stronie [4,8,10,14,21,31,32,33]

W artykule [5,28,34, 15,20]

W książce [27]